04

Modificando la estructura - parte 2

Transcripción

[00:00] Bien. La solución funciona, pero quizás ustedes ya pensaron en una solución más ingeniosa aún, que es ¿qué pasaría si yo vuelvo al funcionario autenticable, del que yo puedo hacer extends de una sola clase y él es un funcionario? ¿Qué te parece si yo hago esto acá? Extends Funcionario. Perfecto.

[00:25] Bueno, me pide implementar métodos, entonces lo que hago es implementar los métodos. Voy aquí, add unimplemented methods, y él ya me generó el getBonificacion, y si FuncionarioAutenticable ya me generó el getBonificacion, ¿qué significa? Que yo puedo sobreescribir los métodos en las clases hijo.

[00:51] Por ejemplo si yo acá le regreso su anotación override, él va a compilar sin problemas porque él ya está sobreescribiendo la clase padre. Con esto, nuestro diagrama quedaría más o menos así: dado que funcionarioAutenticable, vamos a borrar esta referencia de acá, que sigue existiendo pero vamos a ordenar nuestro diagrama primero. Esto de acá lo vamos a bajar un poco. ¿Por qué?

[01:21] Porque está quedando algo así: administrador, aquí está el administrador, y gerente, vamos a quitarle su flechita acá, son funcionarios autenticables, perfecto. Y FuncionarioAutenticable, vamos a copiar aquí una flechita, funcionario autenticable es un funcionario. ¿Conceptualmente correcto? Correcto.

[01:50] Y contador sigue siendo un funcionario, entonces extiende directamente del funcionario y no va a tener pues digamos la característica o la habilidad de poder loguearse en el sistema. Hasta ahora, el sistema está conceptualmente correcto, funciona que es lo más importante, está todo bien. Si vamos a nivel de código compila todo, lo único que no está compilando es aquí la clase CuentaCorriente, porque creo que no hay implementado un método.

[02:26] No ha implementado el método deposita que ya lo implementé ahora y ya está sin problemas. Acá está todo okay. Con esto tenemos nuestros casos de uso completos, tenemos la funcionalidad implementada porque ya lo hemos testeado, y entonces el código funciona, cubre todos los casos de uso y está todo okay, perfecto.

[02:53] Ahora digamos pues que volvemos aquí a nuestro diagrama y ahora vamos a tener un cuarto personaje y es un personaje si no es el más importante es muy importante, vamos a ponerlo por aquí abajo, que es el cliente. ¿Qué es una empresa sin un cliente? ¿Qué es el banco sin los clientes? Que son pues los que dan el dinero, son los que al final pagan las cuentas.

[03:24] Vamos a ponerlo acá. Entonces, ¿qué sería del banco sin el cliente? Ustedes cuando abren su cuenta en el banco, al mismo tiempo dan un acceso a la plataforma del banco. Entonces, vamos a representar aquí es plataforma que va a ser pues nuestro sistema. Aquí nuestro sistema, el sistema interno, vamos a crearlo completo, sistema interno.

[03:59] Ahora, con el sistema interno nosotros tenemos control del administrador, el gerente, y ahora también el cliente puede acceder al sistema del banco. ¿Para qué? Para ver sus saldos o realizar sus operaciones en línea usando nuestro sistema. ¿Sí? Si nosotros sabemos que el administrador, vamos a copiar aquí una flechita para ponerlo un poco más explícito, si el administrador usa el sistema interno para iniciar sesión, el gerente también usa nuestro sistema interno para iniciar sesión.

[04:38] ¿Entonces ustedes creen que el cliente deba usar el sistema interno para iniciar sesión? Debería. ¿Pero cuál es el requisito que pedimos para que sea autenticable? Que sea un funcionario autenticable, entonces ustedes pueden decir: "Ah, perfecto", entonces el cliente puede ser tranquilamente extendemos pues la clase cliente, le creo sus atributos y que extienda de FuncionarioAutenticable. Listo.

[05:08] Y el problema ya está solucionado pues el cliente puede acceder al sistema, realizar sus operaciones y no hay ningún problema, está todo bien. Yo les pregunto a ustedes: ¿El cliente es un funcionario? Comencemos de ese punto de partida. ¿Será que el cliente es un funcionario? Yo creo que no. El funcionario autenticable al mismo tiempo extiende del funcionario, entonces acá tenemos otro detalle.

[05:42] ¿Por qué? Porque el extender de funcionario tiene atributos puntuales como salario, número de cuenta creo que era también, si bien son atributos que el cliente puede tener, no van a ser digamos los mismos atributo en significado, en términos de dominio por así decirlo, son muy diferentes los dominios del funcionario que los del cliente en sí.

[06:11] Entonces si bien satisface la necesidad a nivel de sistema, conceptualmente estamos haciendo una implementación muy, muy errónea. ¿Cuáles van a ser las consecuencias a futuro? La primera consecuencia que puede salir de un modelo mal planteado es el exceso de funcionalidades, porque ustedes pueden decir: "Bueno, tranquilamente no uso los atributos de funcionario y se acabó".

[06:38] De mi cliente sigue extendiendo el FuncionarioAutenticable y lo que no necesito, no lo uso. Okay, eso lo sabes tú. Pero el programador que puede llegar de aquí al siguiente año a darle un mantenimiento al código, ¿lo sabrá? ¿A dos años? ¿A tres años? Hay sistemas bancarios que tienen más de 10 años de antigüedad en su código y tú no sabes si el programador que lo escribió está aquí o está en la China. Es imposible saber, es imposible contactarlo.

[07:09] Entonces, tal como les digo siempre, recuerda que tú escribes código no solo para computadora sino para que sea entendible para otros programadores también. Facilitémonos la vida entre todos. Entonces, volviendo aquí al tema, este modelo de aquí ya está equivocado, ya está errado. ¿Por qué? De repente me dices: "Bueno, ya el cliente no es funcionario, vamos a cambiarle el nombre de la clase, vamos a renombrarlo, vamos a ponerle autenticable". Tenemos solo autenticable.

[07:42] De todas formas autenticable sigue siendo funcionario y el problema aún persiste. Vamos a ver en el siguiente video cómo podemos solucionar este problema y qué otra alternativa tenemos a la herencia, que ya la conocemos muy bien.